# assignment_3_W_23_STUDENT

November 29, 2023

## 1 Assignment 3 (100 marks)

### 1.1 *Generating useful features for further analysis on Amazon reviews*

### 1.2 Introduction

**Business Context.** You are a business consultant with new clients that are interested in analyzing reviews of their products on Amazon (as opposed to Yelp). They want to answer business questions like: "What are the most important factors driving negative reviews?", "Have there been any large changes to customer satisfaction/reviews over time?", etc.

**Business Problem.** Your main task is to **explore the given data and use the results of your investigation to engineer relevant features that could facilitate subsequent analysis and model-building**.

**Analytical Context.** The dataset provided is a large body of reviews related to movies and television left on Amazon between 1996 and 2014. When exploring our dataset, we will quickly encounter a familiar problem we discussed in the previous case: the word "good" is one of the most important words in both positive *and* negative reviews. Thus, we must develop methods to put "good" in the appropriate context.

### 1.3 Loading the data

We use a dataset of around 37,000 video reviews from Amazon Instant Video and 1,700,000 movie and TV reviews, all obtained from the website: http://jmcauley.ucsd.edu/data/amazon/. Note that there are much larger datasets available at the same site. We can expect better and more consistent results on larger datasets (such as book reviews). Note that these datasets are compressed (gzipped), and they are in JSON format, with each line representing a review and each line being its own JSON object.

We begin by loading the dataset below:

[893]:
```
%%time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels
import statsmodels.api as sm
from statsmodels.formula.api import ols
from scipy import stats
```

```
from pingouin import pairwise_tests #this is for performing the pairwise tests
from pingouin import pairwise_ttests #this is for performing the pairwise tests
from sklearn.feature_extraction.text import CountVectorizer
from datetime import datetime
import gzip
import json
import string
import nltk # imports the natural language toolkit
import plotly

%matplotlib inline
nltk.download('punkt')

nltk.download('stopwords')
from nltk.corpus import stopwords
# We won't use this one this time
# we can tell pandas that our file is in gzip format and it will handle the␣
 ↪decompression for us
# we also use `lines=True` to indicate that each line of the file is its own␣
 ↪JSON object
# instant_video = pd.read_json("reviews_Amazon_Instant_Video_5.json.gz",␣
 ↪lines=True, compression='gzip')



# ----------
# The Movies and TV file is very big. If you have problems loading it, you can␣
 ↪load only the first
# 100,000 reviews by using 'chunksize' (uncomment the line with 'chunksize' and␣
 ↪comment out the line
# after that which loads the entire file into `movies_tv`). All of the analysis␣
 ↪can be
# done in the same way using only the subset of reviews but some of the results␣
 ↪might be different from the examples.
# ----------
movies_tv = next(pd.read_json("reviews_Movies_and_TV_5.json.gz", lines=True,␣
 ↪compression='gzip', chunksize=100000))
#movies_tv = pd.read_json("reviews_Movies_and_TV_5.json.gz", lines=True,␣
 ↪compression='gzip')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /home/88357a33-1bce-4a0e-9ccf-d3051f3ef0f0/nltk_data..
[nltk_data]     .
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/88357a33-1bce-4a0e-9ccf-d3051f3ef0f0/nltk_data..
[nltk_data]     .
[nltk_data]   Package stopwords is already up-to-date!
```

```
CPU times: user 2.5 s, sys: 212 ms, total: 2.72 s
Wall time: 2.72 s
```

### 1.4 Examining the data

We take a look at the first 5 rows of each dataset to see what attributes are available. These are

- **reviewerID:** A unique ID to identify the author of the review.
- **asin:** The "Amazon Standard Identification Number" which provides more information about the exact product and version.
- **reviewerName:** The username chosen by the reviewer.
- **helpful:** A record of how many users indicated that the review was helpful/not helpful.
- **reviewText:** The full text of the review.
- **overall:** The overall rating (1-5) left by the reviewer.
- **summary:** A short version of the review, used as the title.
- **unixReviewTime:** The date that the review was created, in Unix Epoch format.
- **reviewTime:** A human readable date giving the day, month, and year.

[895]:
```
print(len(movies_tv))
print(movies_tv.head(5))
```

```
100000
       reviewerID        asin                          reviewerName helpful  \
0   ADZPIG9QOCDG5  0005019281  Alice L. Larson "alice-loves-books"  [0, 0]
1  A35947ZP82G7JH  0005019281                         Amarah Strack  [0, 0]
2  A3UORV8A9D5L2E  0005019281                      Amazon Customer  [0, 0]
3  A1VKW06X1O2X7V  0005019281           Amazon Customer "Softmill"  [0, 0]
4  A3R27T4HADWFFJ  0005019281                                BABE  [0, 0]

                                          reviewText  overall  \
0  This is a charming version of the classic Dick…        4
1  It was good but not as emotionally moving as t…        3
2  Don't get me wrong, Winkler is a wonderful cha…        3
3  Henry Winkler is very good in this twist on th…        5
4  This is one of the best Scrooge movies out.  H…        4

                                    summary  unixReviewTime   reviewTime
0                   good version of a classic      1203984000  02 26, 2008
1                      Good but not as moving      1388361600  12 30, 2013
2        Winkler's Performance was ok at best!      1388361600  12 30, 2013
3  It's an enjoyable twist on the classic story      1202860800  02 13, 2008
4                             Best Scrooge yet      1387670400  12 22, 2013
```

We notice that `movies_tv` is extremely long with nearly 2 million reviews, and several columns seem uninteresting or hard to work with (e.g. `reviewerID`, `asin`, `reviewername`, `reviewtime`). We drop some information to make some of our later analysis more efficient. We also add a datetime column with Python datetime objects to more easily summarize the data:

[897]:
```
movies_tv['datetime'] = pd.to_datetime(movies_tv['reviewTime'], format="%m %d,␣
↪%Y")
```

[898]:
```
movies_tv = movies_tv.drop(columns = ['reviewerID', 'asin', 'reviewerName',␣
↪'reviewTime'])

movies_tv.head(5)
```

[898]:

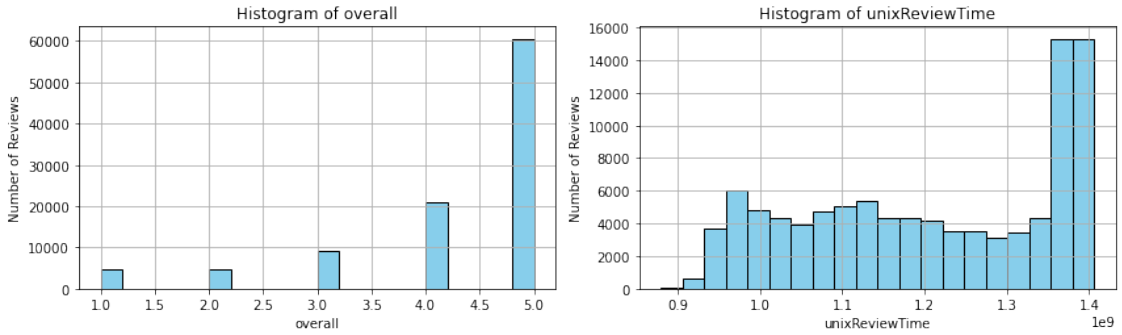
| | helpful | reviewText | overall | summary | unixReviewTime | datetime |
|---|---|---|---|---|---|---|
| 0 | [0, 0] | This is a charming version of the classic Dick… | 4 | good version of a classic | 1203984000 | 2008-02-26 |
| 1 | [0, 0] | It was good but not as emotionally moving as t… | 3 | Good but not as moving | 1388361600 | 2013-12-30 |
| 2 | [0, 0] | Don't get me wrong, Winkler is a wonderful cha… | 3 | Winkler's Performance was ok at best! | 1388361600 | 2013-12-30 |
| 3 | [0, 0] | Henry Winkler is very good in this twist on th… | 5 | It's an enjoyable twist on the classic story | 1202860800 | 2008-02-13 |
| 4 | [0, 0] | This is one of the best Scrooge movies out. H… | 4 | Best Scrooge yet | 1387670400 | 2013-12-22 |

### 1.4.1 Exercise 1 (22 marks):

**1.1 (4 marks)** Plot histograms of all numeric quantities for the `movies_tv` DataFrame. Note any observations (a few bullet points is fine).

[900]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Assuming 'movies_tv' DataFrame is already prepared as mentioned in the context

# Selecting only numeric columns for histogram plotting
numeric_columns = movies_tv.select_dtypes(include=['int64', 'float64']).columns.
↪tolist()

# Plot histograms for numeric columns
plt.figure(figsize=(12, 10))
for i, col in enumerate(numeric_columns, 1):
    plt.subplot(3, 2, i)
    plt.hist(movies_tv[col], bins=20, color='skyblue', edgecolor='black')
    plt.xlabel(col)
    plt.ylabel('Number of Reviews')
    plt.title(f'Histogram of {col}')
    plt.grid(True)

plt.tight_layout()
```

```
plt.show()
print("Note any observations (a few bullet points is fine).\n");

# Splitting 'helpful' column into two separate columns
movies_tv[['helpful_count', 'not_helpful_count']] = pd.
  ↪DataFrame(movies_tv['helpful'].tolist())
print(movies_tv)
```




```
Note any observations (a few bullet points is fine).

      helpful                                         reviewText  overall  \
0      [0, 0]  This is a charming version of the classic Dick…        4
1      [0, 0]  It was good but not as emotionally moving as t…        3
2      [0, 0]  Don't get me wrong, Winkler is a wonderful cha…        3
3      [0, 0]  Henry Winkler is very good in this twist on th…        5
4      [0, 0]  This is one of the best Scrooge movies out.  H…        4
…        …                                                 …      …
99995  [0, 0]  If there is one Joan Crawford movie to own on …        5
99996  [1, 1]  Bette Davis and Joan Crawford were two over th…        5
99997  [1, 3]  I love this film, It is like a heavyweight fig…        5
99998  [1, 1]  I'm glad this was in black and white because o…        5
99999  [2, 4]  Liz Taylor and Richard Burton team up in Edwar…        5

                                        summary  unixReviewTime  \
0                         good version of a classic      1203984000
1                            Good but not as moving      1388361600
2               Winkler's Performance was ok at best!      1388361600
3        It's an enjoyable twist on the classic story      1202860800
4                                  Best Scrooge yet      1387670400
…                                              …            …
99995                                   Brilliant      1056153600
99996                        This is a classic…       1065225600
99997            Bette and Joan. Cinema Heavyweights      1357516800
99998                                        Misery      1221091200
99999                          Intense Psychodrama       944956800
```

```
        datetime  helpful_count  not_helpful_count
0     2008-02-26              0                  0
1     2013-12-30              0                  0
2     2013-12-30              0                  0
3     2008-02-13              0                  0
4     2013-12-22              0                  0
…            …              …                  …
99995 2003-06-21              0                  0
99996 2003-10-04              1                  1
99997 2013-01-07              1                  3
99998 2008-09-11              1                  1
99999 1999-12-12              2                  4

[100000 rows x 8 columns]
```

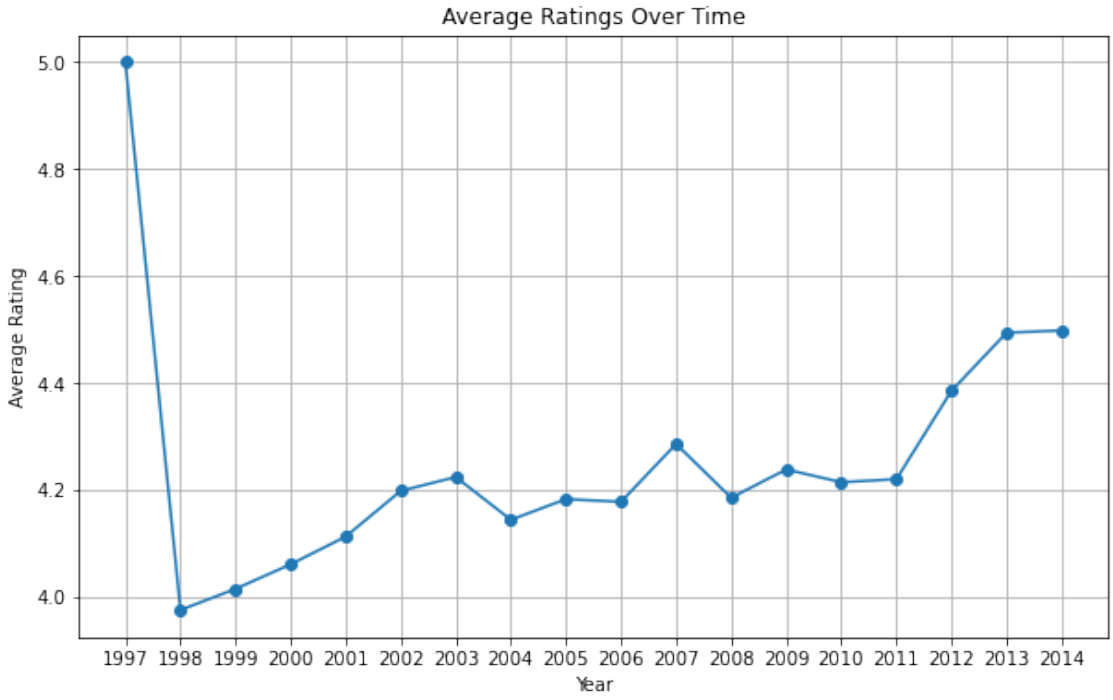
[901]:
```
movies_tv = movies_tv.drop(columns = ['helpful_count', 'not_helpful_count'])
```

- The first graph that was created based on the numeric values from the movies_tv dataframe, is the "Histogram of overall". This is a histogram that displays the overall star ratings for all of the reviews that were recorded. The number of reviews is on the y-axis, whereas the actual star rating is shown on the x-axis. For example, we can see that 5 stars is the most common overall rating, and more than 60,000 reviews included a 5 star rating for the movie. In this graph, I also noticed that the least frequent rating is 1 and 2 stars, and these 2 ratings have a very similar number of corresponding reviews.
- The second graph that was created based on the numeric values from the movies_tv dataframe, is the "Histogram of unixReviewTime". This is a histogram that represents the time when a review was submitted, measured in Unix time. Unix time is basically a system for tracking time as a number of seconds that have elapsed from a specific start date and it's a way to represent timestamps in a standardized format. From the graph, we can see that the unixReviewTimes from approximately 13.5x10^9 secs - 14x10^9 have enormously more reviews when compared to any other timestamp. This means that this is the time period where most reviews for movies were posted, in a short amount of time. Around the unixReviewTime of 0.9x10^9 secs, we can see that thew number of reviews is very low when compared to the rest (well below 1000). So, we can conclude from this that the dataframe started to first collect data from around this time period/timestamp (0.9x10^9 secs).

**1.2 (4 marks)** How do average ratings change over time in the `movies_tv` DataFrame? Plot the average rating for each year and note any trends (a few bullet points is fine).

[904]:
```
# Extracting year from the 'datetime' column and converting to integer
movies_tv['year'] = movies_tv['datetime'].dt.year.astype(int)

# Calculating average rating for each year
average_ratings = movies_tv.groupby('year')['overall'].mean().reset_index()

# Plotting average ratings over time with years as integers on the x-axis
```

```
plt.figure(figsize=(10, 6))
plt.plot(average_ratings['year'], average_ratings['overall'], marker='o',␣
↪linestyle='-')
plt.xlabel('Year')
plt.ylabel('Average Rating')
plt.title('Average Ratings Over Time')
plt.xticks(average_ratings['year'])  # Set x-axis ticks as years
plt.grid(True)
plt.show()
print("Note any trends (a few bullet points is fine).");
```




Note any trends (a few bullet points is fine).

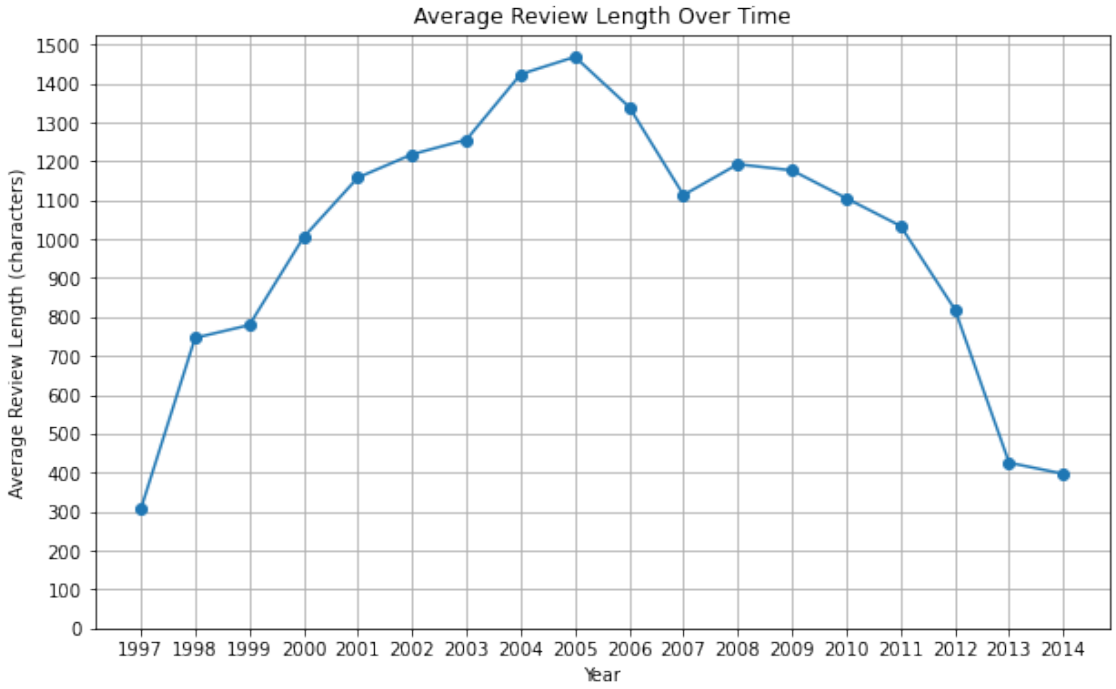
- The first trend that I notice is that 1997 starts off with a very high overall rating of 5 stars.
- After 1997, there is a significant drop in the average movie rating (overall) in 1998, as it drops below 4 stars. I think that the main reason for this, is that there is more data to work with.
- From 1998 to 2003, we can see a slight increase in the average rating as each year goes by.
- In 2007, the rating is increased by a good amount but in 2008, it decreases by roughly the same amount again.
- Then, from 2010-2014, we can see a consistent trend of the increasing yearly rating again (especially a decent jump from 2011-2012).

[906]:
```
movies_tv = movies_tv.drop(columns = ['year'])
```

**Answer.**

**1.3 (4 marks)** Plot the average length of the reviews in the `movies_tv` DataFrame for each year. Note any trends (a few bullet points is fine).

[909]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'movies_tv' DataFrame is already prepared with 'datetime' column

# Calculate the length of each review text and add it as a new column
movies_tv['review_length'] = movies_tv['reviewText'].str.len()

# Extracting year from the 'datetime' column and converting to integer
movies_tv['year'] = movies_tv['datetime'].dt.year.astype(int)

# Calculate the average review length for each year
average_review_length = movies_tv.groupby('year')['review_length'].mean().
 ↪reset_index()

# Plotting average review length over time
plt.figure(figsize=(10, 6))
plt.plot(average_review_length['year'], average_review_length['review_length'],␣
 ↪marker='o')
plt.xlabel('Year')
plt.ylabel('Average Review Length (characters)')
plt.title('Average Review Length Over Time')
plt.xticks(average_review_length['year'])  # Set x-axis ticks as years
plt.yticks(range(0, int(average_review_length['review_length'].max()) + 100,␣
 ↪100))  # Set y-axis ticks in increments of 100
plt.grid(True)
plt.show()
```




- From 1997-1998, we can see an increase in average length by roughly 450 characters, which is a pretty big increase.
- The reviews continue to increase in average length each year, up until 2005.
- From 2005-2007, the graph experiences its very first trend of decrease in the average review length.
- There is a slight increase in average review length from 2007-2008, however this doesn't stay consistent since from 2008-2014, the average review length over the years continues in the downwards trend.
- From these trends, we might conclude that during the earlier years, the reviewers may have been excited to write the reviews and had lots of opinions, so they tried to write more and more as the years went by. However, in 2005, this trend was basically flipped as the average reviewer began writing less and less as years went by.
- The shape of the graph reminds me of a parabola (but less consistent).

**1.4 (10 marks)** We saw that movies and TV seem to have higher ratings in the 90s. Use a hypothesis test to determine if this is a statistically significant pattern. Specifically, - State the null and alternative hypotheses - Run the appropriate hypothesis test in python - Use the output to decide if you reject or fail to reject the null hypothesis - Interpret the p-value in this context

**Answer.**

[913]:
```
#the null hypothesis will always correspond to the hypothesis of no change.

ratings_90s = average_ratings[(average_ratings['year'] >= 1997) &␣
 ↪(average_ratings['year'] <= 1999)]['overall']
```

```
ratings_other = average_ratings[(average_ratings['year'] >= 2000) &␣
 ↪(average_ratings['year'] <= 2014)]['overall']

mean_ratings_90s= ratings_90s.mean()

ratings_other_mean= ratings_other.mean()
print("Null Hypothesis ( 0): The average ratings of movies and TV shows in the␣
 ↪90s are the same as the average ratings in other periods (not in the 90s).")
print(" 0 = "+str(mean_ratings_90s))
print(" 0:   =  0")
print(" 0:   = "+str(mean_ratings_90s))

print("\nAlternative hypothesis ( ): The average ratings of movies and TV shows␣
 ↪in the 90s are significantly higher from the average ratings in other␣
 ↪periods.")
# : <mean_ratings_90s --> avg ratings of movies in other periods are less than␣
 ↪ratings during the 90s.
print(" a:   <  0")
print(" a:   < "+str(mean_ratings_90s))
#ratings_other_mean


# Performing a hypothesis test -->  < 4.330001921968095,   = ratings_other
stat, p_value =stats.ttest_1samp(ratings_other, mean_ratings_90s,␣
 ↪alternative='less')

# Printing the results
print("\nStatistic:", stat)
print("P-Value:", p_value)

#Use the output to decide if you reject or fail to reject the null hypothesis␣
 ↪--> p-value
alpha = 0.05  # desired significance level
if p_value < alpha:
    print("Reject the null hypothesis.")
    print("There is evidence to suggest that the average ratings in the 90s are␣
 ↪significantly higher than other periods.")
else:
    print("Fail to reject the null hypothesis.")
    print("There is not enough evidence to conclude that the average ratings in␣
 ↪the 90s are significantly higher from other periods.")
```

```
Null Hypothesis ( 0): The average ratings of movies and TV shows in the 90s are
the same as the average ratings in other periods (not in the 90s).
 0 = 4.330001921968095
 0:   =  0
 0:   = 4.330001921968095
```

```
Alternative hypothesis ( ): The average ratings of movies and TV shows in the
90s are significantly higher from the average ratings in other periods.
 a:   < 0
 a:   < 4.330001921968095

Statistic: -2.705908530363807
P-Value: 0.008529049147588045
Reject the null hypothesis.
There is evidence to suggest that the average ratings in the 90s are
significantly higher than other periods.
```

Interpretation of p-value:

Since the p-value in this case is less than alpha (the desired significance level/threshold), we can reject the null hypothesis. What this means, is that since the condition of p_value<alpha is met, there is enough evidence for us to conclude that alternative hypothesis can be accepted, and that the null hypothesis can be rejected. So, this means that it is valid to believe that the average overall ratings in the 90s are significantly higher than other periods.

### 1.4.2 Exercise 2 (10 marks):

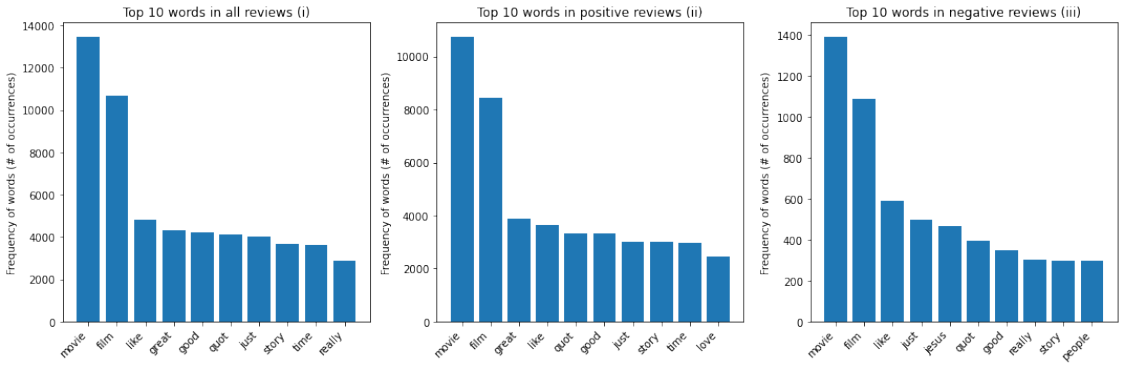
For the remainder of the assignment, we will use a smaller version of the `movies_tv` DataFrame given below:

[916]:
```
short_movies_tv = movies_tv.head(10000)
```

Find the ten most frequently occurring non-stop words across: (i) all reviews, (ii) positive reviews (higher than 3 stars), (iii) (lower than 3 stars) negative reviews. Make a bar plot of the ten most frequently occurring words against their frequencies for each category ((i) (ii) and (iii)). What do you notice when comparing the graphs? Do the results surprise you? Why or why not? (A few bullet points is fine)

[918]:
```
# Assume 'short_movies_tv' is your DataFrame containing 'reviewText' and␣
 ↪'stars' columns

# Define a function to get the top n non-stop words
def get_top_non_stop_words(corpus, n=1,k=1):
    vec = CountVectorizer(ngram_range=(k,k),stop_words = 'english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.
 ↪items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

# Get all reviews, positive reviews, and negative reviews
#if 3 stars -- good review
all_reviews = short_movies_tv['reviewText']
```

```
positive_reviews = short_movies_tv[short_movies_tv['overall'] > 3]['reviewText']
negative_reviews = short_movies_tv[short_movies_tv['overall'] < 3]['reviewText']

# Get top non-stop words for each category
top_all_words = get_top_non_stop_words(all_reviews, 10, 1)

top_positive_words = get_top_non_stop_words(positive_reviews, 10,1)
top_negative_words = get_top_non_stop_words(negative_reviews, 10, 1)


# Bar plots for the ten most frequently occurring non-stop words
plt.figure(figsize=(15, 5))

plt.subplot(131)
plt.bar([word[0] for word in top_all_words], [word[1] for word in␣
 ↪top_all_words])
plt.title('Top 10 words in all reviews (i)')
plt.ylabel('Frequency of words (# of occurrences)')
plt.xticks(rotation=45, ha='right')

plt.subplot(132)
plt.bar([word[0] for word in top_positive_words], [word[1] for word in␣
 ↪top_positive_words])
plt.title('Top 10 words in positive reviews (ii)')
plt.ylabel('Frequency of words (# of occurrences)')
plt.xticks(rotation=45, ha='right')

plt.subplot(133)

#negative_words = [word[0] for word in top_negative_words]
#negative_frequencies =[word[1] for word in top_negative_words]\

plt.bar([word[0] for word in top_negative_words], [word[1] for word in␣
 ↪top_negative_words])
plt.title('Top 10 words in negative reviews (iii)')
plt.ylabel('Frequency of words (# of occurrences)')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
plt.tight_layout()
plt.show()
```




- The most obvious thing that I notice is that the first two graphs (i) and (ii), have the exact same words in the top 10. The reasoning behind this is that most of the reviews are above 3 stars, and are considered as "good reviews". Since the proportion of good reviews is a lot larger than the proportion of bad reviews (more good data to work with), the top words in the bad reviews are essential overshadowed by the top words in good reviews, when looking at top 10 words in all reviews.
- I am not surprised by this since the frequency/range difference in graphs (ii) and (iii) also tells us how much lower of an impact the bad reviews really have.
- Also, I believe the word 'quot' actually represents the quotation character that comes up in the reviews. This makes sense, since during movie reviews, lots of people might be mentioning quotes from the actual movie.

### 1.4.3 Exercise 3 (8 marks):

Find 20 words that are indicative of bad reviews. That is, come up with a method to identify words that appear frequently in the bad reviews but do *not* occur frequently in the good reviews. What are these words? What do you observe about these words? Are they surprising?

[921]:
```
from collections import Counter
from nltk.tokenize import word_tokenize


# Define a function to get the top n non-stop words
def get_top_non_stop_words(corpus, n=1,k=1):
    vec = CountVectorizer(ngram_range=(k,k),stop_words = 'english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.
 ↪items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
#n = 77, cuz its much bigger than 20
```

```
top_positive_words = get_top_non_stop_words(positive_reviews, 77,1) #i chnaged␣
 ↪this so thaat it is not fixated on top 20 words!
top_n_negative_words = get_top_non_stop_words(negative_reviews,77, 1)

# Identify words frequent in bad reviews but infrequent in good reviews
indicative_words = [word[0] for word in top_n_negative_words]
#print(indicative_words)
# Check and remove words if they're in positive reviews with an occurence > 10␣
 ↪from top n reviews.
indicative_words = [word for word in indicative_words if not any(pos_word[0] ==␣
 ↪word and pos_word[1] > 10 for pos_word in top_positive_words)]
print('\nWords that are indicative of bad reviews: ')
print(indicative_words)
print("\nNumber of words shown above: "+ str(len(indicative_words)) )
```

```
Words that are indicative of bad reviews:
['bad', 'didn', 'violence', 'mel', 'jane', 'god', 'plot', 'thing', 'thought',
'doesn', 'going', 'believe', 'actually', 'point', 'things', 'got', 'money',
'big', 'fact', 'far', 'guy', 'instead']

Number of words shown above: 22
```

Something I observe about these 20 words is that there is a lot of comparing/contrasting in the bad reviews. For example, words like "instead", "far", and "actually" could potentially indicate a contrast between expectations and reality, implying that the viewer was disappointed. Also, the words "mel" and "jane" were surprising to me, however I think that they could be names of characters from a specific movie that has a very low rating, and hence many bad reviews. There are also many words with negative connotations such as "didn't", "violence", and "doesn't".

### 1.4.4 Exercise 4 (8 marks):

Use the following code to inspect the first 50 1-star reviews from the `movies_tv` DataFrame containing the word "good". Discuss the most common ways the word good is used in these reviews. Are any of these informative or challenging for our analysis of what drives very negative reviews?

[924]:
```
v_bad_indices = movies_tv['overall'] < 1.1
v_bad_reviews = movies_tv[v_bad_indices]

RED_START = "\033[91m"
END = "\033[0m"

good_word_bad_review = [rev for rev in v_bad_reviews['reviewText'] if 'good' in␣
 ↪rev.lower()]

print(len(good_word_bad_review))
for rev in good_word_bad_review[:50]: #extract first 50 1 star reviews
    good_i = rev.lower().index("good")
```

```
    # we add Ansi codes to highlight the matches
    print(rev[good_i-50: good_i] + RED_START + rev[good_i:good_i+4] + END +␣
↪rev[good_i+4:good_i +50])
```

```
1266
good, the story was correct, but naked backsides w
good but the scene with Yukon discovering peppermi
th absolutely NO edits whatsoever to the material.Good for us!Edits are
for losers!I have spoken.
t least made it about an interesting story with a good plot and
character development…i agree with
t everything I've seen thus far does not resemble good cinema in any
shape, form, or manner. So, I g
itself is beautifully filmed. The acting is quite good and as a history
freak I found it interesting
rtaining) and provides nothing that makes us feel good or is uplifting
or is of any benefit, except
ven if that person is Christ. The flashbacks were good and I'd rather
have watched them than watched
to explain what singled out Christ as a symbol of good because a)he is
a shallow and poor filmmaker
 (and no-one could endure the pain).They say it's good to show the
extent of the sacrifice that Jesu
good review when it was released, and I went to se
 story that everyone knows and of corse it's very good and
interesting.I'm sorry, lots of people mus
into the character's pain and visceral anguish is good film making but
the level to which Mel Gibson
the guitar riff, so at least I could have enjoyed good music.  By the
way, The Passion's music sucks
Good grief, this movie was painful to watch and di
of Jesus?  Because, my friend, that simply is not good box office.
Whaddre we making here, some kin
t his head off either.) but doesn't show that any good comes out of
it.Fifth: The Devil. What's with
 proves it. If you enjoyed this movie, there is a good chance you have
an undiagnosed mental illness
ation;  I left the theater thinking "Was that any good?"  After a while
it hit me; after this film w
good actor and a good director, when he just makes
ing else in this film.The scourging of Jesus is a good example of
Gibson's contempt for facts.  Roma
rth. I have to say he did not do anywhere near as good a job as
Jackson. And I'm totally lost as to
't think that anyone in this film got off looking good. Including Jesus
himself.*The mob of people r
he sacrifice that was made for mankind. There are good depictions of
```

the life and sacrifice of Chris
nd of corporate Christendom and sold as a bill of goods that takes us
to a "new level in realism" be
Ok, that's understandable… it just doesn't make good cinema.My other
problem with the movie are th
HE REVIEW ON AMAZON, SUGGESTED IT MIGHT BE PRETTY GOOD, BUT I HAVE
VIEWED OVER 240 CHRISTMAS MOVIES
good one at that.  I can take my old VHS tape and
g. It's not flat out terrible, but it's far from "good".EXTRA
FEATURES:Seriously, what did you expec
enes (most of the film) the picture blurs!The old Goodtimes LP speed
VHS tape from the 80s looks bet
in a novel?The movie, on the other hand, isn't so good.From the very
first scene you get the feeling
ntly restores the picture quality as well as adds good special features
and has good audio options s
 fine, quick shipping and decent packing but what good is that if the
item is not usable in this cou
must be an absolutely humorless person.  The only good thing that can
be said is that this failure f
 movie at all.  Most of what makes the stories so good is his thoughts
& reactions to what is going
 ten dollars for a playing of condor man.  It's a good movie but not
that good.  Besides I have acop
lm becomes absolutely tiresome to watch. The only good things about it
are seeing Jennifer Connelly
her customer reviews I thought it was going to be good. A wasted thirty
minutes of my life.
Good Movie, not great, the Dubbing is what bothere
at it was  ruined by the fact that it was dubbed. Good story though!
of tea this is your movie.  Also, there is a very good hidden message
about,"Do movies make peo
when it first came out and for some reason it had good reviews so I
thought I would give it a chance
the acting is excellent, but in order to create a good movie you
absolutely must start with a good s
 the very end. Maybe the last 10 minutes or so is good. I love
Christmas movies, and have seen almos
lling suspension of disbelief" that is key to any good narrative.
Besides, in this day and age, mov
ey decided to focus on the bad guy instead of the good guy - a huge
mistake. Without a protagonist t
ted to marry me." again, the self-satisfied smile.Good lord! some
advice: rent or buy either the 194
antha Morton and Ciaràn Hinds they had two good actors, who even
physically fit their roles w
good version of Jane Eyre except for the terrible
this movie  seems very rushed you can never get a good foot hold of the

`story Especially during Jane`

**Answer.**

I have observed a few different ways the word "good" is used in these 1-star reviews. The first way I see it being used is in the form of negative connotations. For example, "good grief", "good lord", or "not that good" are some phrases from the reviews and they convey disappointment, even though they start with the positive word "good". The word "good" is also used when the reviewers talk about what makes a good movie and how the movie that they watch did not meet those requirements, or used when referring to other improvements that the movie should have. Some examples of this use are "to create a good movie you absolutely must start…", "is good film making but the level to which", and "so at least I could have enjoyed good music…". Some reviews with the word "good" also show mixed sentiments. For example, it has expressions like "it's not flat out terrible, but it's far from"good" ","Good story though¡', and"The flashbacks were good and I'd rather have watched them".

I think that the usage of "good" in these reviews does pose some challenges when analyzing what drives negative reviews. This is because the word is used in diverse contexts, ranging from mixed opinions to direct criticism. For example, the two phrases "good story though!" and "the story was not that good", both use the word "good" but have a different effect, tone, and meaning. So, interpreting the sentiment behind the word "good" can become very intricate, as it usually requires a deeper understanding of the reviewer's context, expectations, and reasons as to why they are not satisfied.

Something informative from these reviews is that 1-star reviews usually have lots of comparisons since the viewer's expectations may not have been met. So, the word "good" is often used in this context (ex: "not as good") and it drives comparisons in negative reviews.

### 1.4.5 Exercise 5 (15 marks):

For each review in the list of bad reviews containing the word "good" that we found in the last question (`good_word_bad_review`) extract the following:

1. The first word after "good"
2. The first word after "good" that is a noun or cardinal
3. The last word before "good" that is a noun or cardinal

For example, "The popcorn at the movie was very good with butter." would have the solution

1. with
2. butter
3. movie

Print out the results for the first 15 reviews in the list `good_word_bad_review`.

**Answer.**

[929]:
```
first_word_after_good=[]

# Function to check if a word is a noun or cardinal number
def is_noun_or_cardinal(word):
    good_pos = ['NN', 'CD'] #noun or cardinal
```

```
    words = nltk.word_tokenize(word)
    pos_tags = nltk.pos_tag(words)
   # return pos[0][1] in ['NN', 'NNS', 'NNP', 'NNPS', 'CD']
    noun_or_cardinal = [k for k,v in pos_tags if v in good_pos]# Checking for␣
 ↪nouns or cardinal numbers
    return noun_or_cardinal

count = 1
print("\nReview "+str(count)+": \n")
# Iterate through the bad reviews containing 'good'
for review in good_word_bad_review[:15]:

    good_index = review.lower().find('good')

    # If 'good' is found in the review
    if good_index != -1:
        # Split the review after 'good'
        words_after_good = review[good_index + len('good'):].split()
        words_before_good = review[:good_index].split()
        words_before_good.reverse() # to get "last" noun/cardianl before good

        # Extract the first word after 'good' if it exists
        if len(words_after_good) > 0:
         #   first_word_after_good.append(words_after_good[0])
            print(f"1. "+words_after_good[0])

        # Extract the first noun or cardinal after 'good'
        for word in words_after_good:
            if is_noun_or_cardinal(word):
              #  print(f"2. {word} ({tag})")
                print(f"2. {word}") #chose this way to output to match solution␣
 ↪example given
                break

        # Extract the last noun or cardinal before 'good'
        for word in words_before_good:
            if is_noun_or_cardinal(word):
                print(f"3. {word}")
                break
        count+=1;
        if(count!=16):
            print("\nReview "+str(count)+": \n")
```

```
Review 1:

1. ,
```

2. story

Review 2:

1. but
2. scene
3. transfer

Review 3:

1. for
2. spoken.
3. material.

Review 4:

1. plot
2. plot
3. story

Review 5:

1. cinema
2. cinema
3. everything

Review 6:

1. and
2. history
3. movie

Review 7:

1. or
2. benefit,
3. feel

Review 8:

1. and
2. violent
3. Christ.

Review 9:

1. because
2. shallow

```
3. symbol

Review 10:

1. to
2. show
3. pain).They

Review 11:

1. review
2. review
3. film

Review 12:

1. and
2. interesting.I'm
3. corse

Review 13:

1. film
2. film
3. anguish

Review 14:

1. music.
2. music.
3. riff,

Review 15:

1. grief,
2. grief,
```

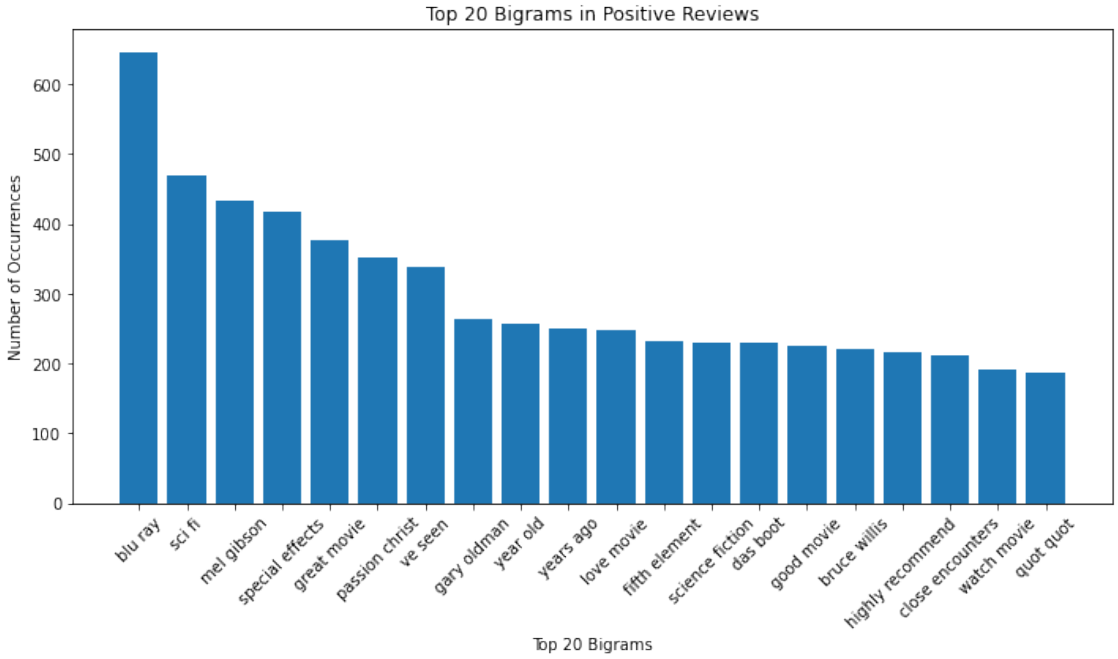
### 1.4.6 Exercise 6 (20 marks):

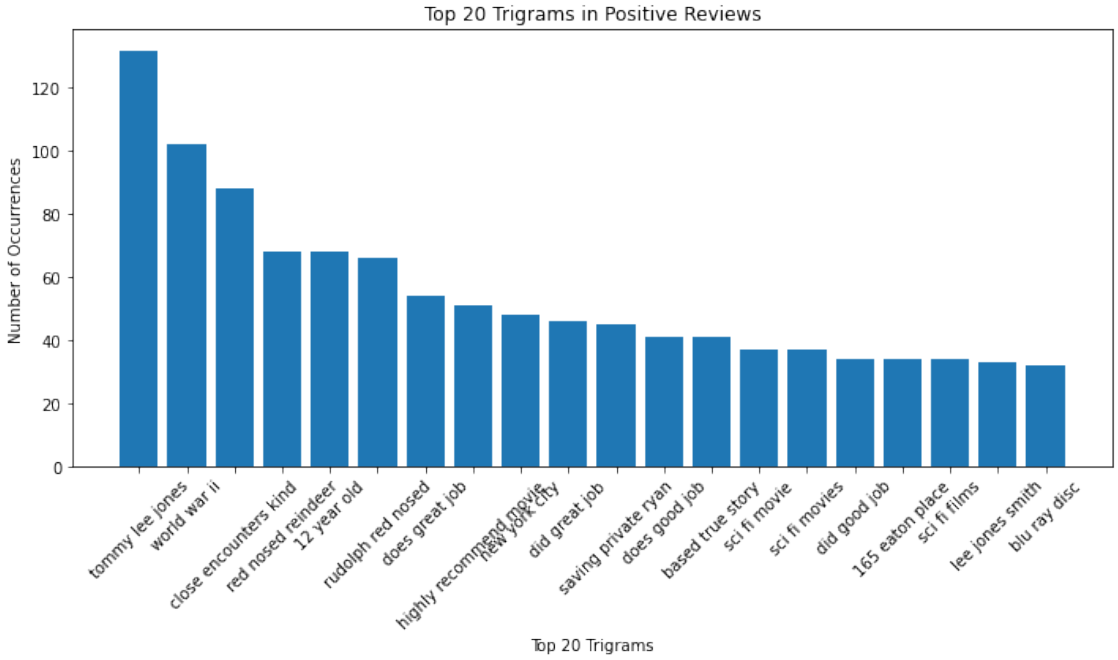
We have seen that individual words are not always very informative. Find the 20 most often occurring bigrams and trigrams in the (i) positive and (ii) negative reviews. Visualize the most frequently occurring bigrams and trigrams in (i) and (ii) and give a brief analysis of the n-grams you identified.

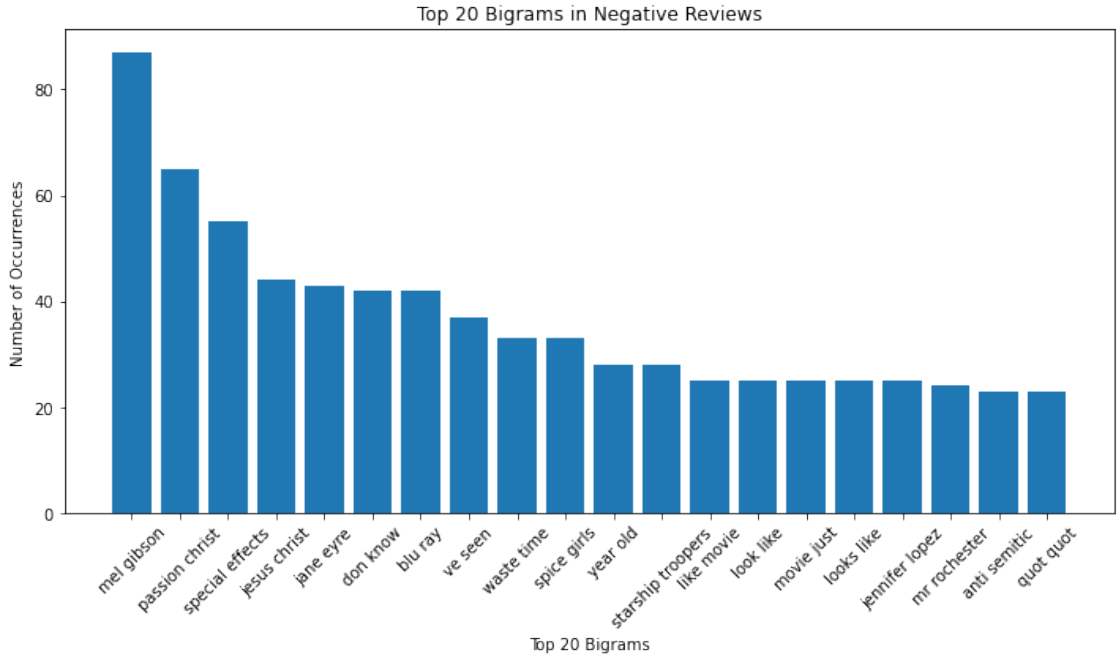
**Answer.**

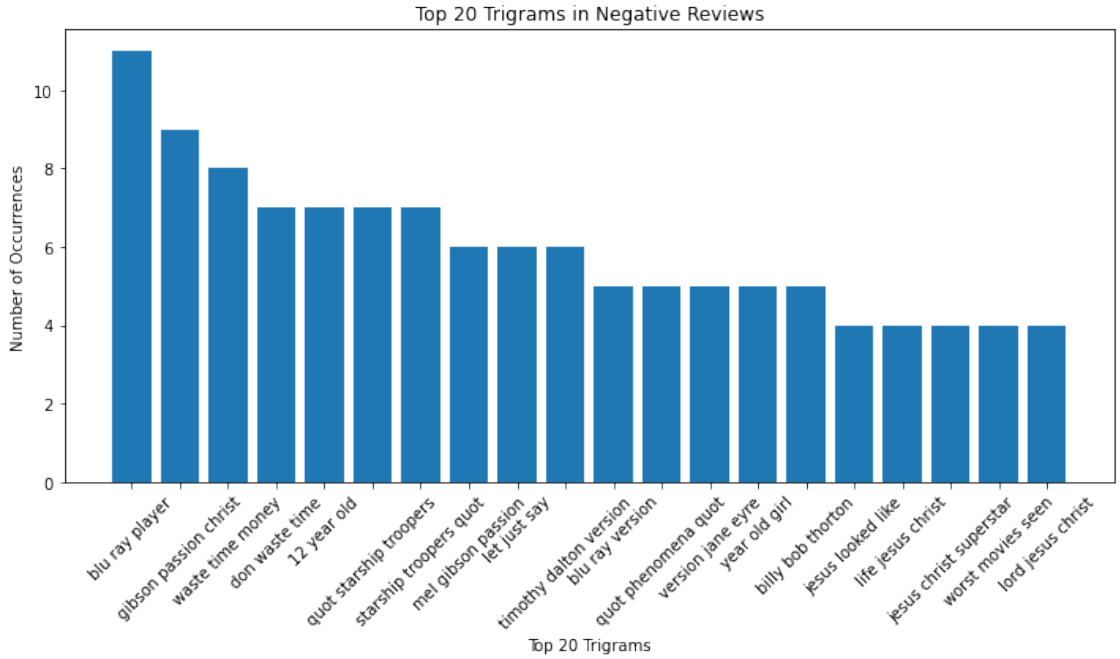
[932]:
```python
def visualize_ngrams(common_words, title, Ngram):
    df = pd.DataFrame(common_words, columns=[Ngram, 'count'])
```

```
    plt.figure(figsize=(10, 6))
    plt.bar(df[Ngram], df['count'])
    plt.xticks(rotation=45)
    plt.title(title)
    plt.ylabel('Number of Occurrences')
    plt.xlabel(Ngram)
    plt.tight_layout()
    plt.show()

# Get top 20 bigrams and trigrams for positive reviews (i)
top_20_positive_bigrams = get_top_non_stop_words(positive_reviews, n=20, k=2)␣
 ↪#bigram
top_20_positive_trigrams = get_top_non_stop_words(positive_reviews, n=20, k=3)␣
 ↪#trigram

# Get top 20 bigrams and trigrams for negative reviews (ii)
top_20_negative_bigrams = get_top_non_stop_words(negative_reviews, n=20, k=2)␣
 ↪#bigram
top_20_negative_trigrams = get_top_non_stop_words(negative_reviews, n=20, k=3)␣
 ↪#trigram

# Visualize top bigrams and trigrams for positive and negative reviews
visualize_ngrams(top_20_positive_bigrams, 'Top 20 Bigrams in Positive Reviews',␣
 ↪'Top 20 Bigrams')
visualize_ngrams(top_20_positive_trigrams, 'Top 20 Trigrams in Positive␣
 ↪Reviews', 'Top 20 Trigrams')
visualize_ngrams(top_20_negative_bigrams, 'Top 20 Bigrams in Negative Reviews',␣
 ↪'Top 20 Bigrams')
visualize_ngrams(top_20_negative_trigrams, 'Top 20 Trigrams in Negative␣
 ↪Reviews', 'Top 20 Trigrams')
```


Top 20 Bigrams in Positive Reviews
Number of Occurrences
0
100
200
300
400
500
600
blu ray
sci fi
mel gibson
special effects
great movie
passion christ
ve seen
gary oldman
year old
years ago
love movie
fifth element
science fiction
das boot
good movie
bruce willis
highly recommend
close encounters
watch movie
quot quot
Top 20 Bigrams


Top 20 Trigrams in Positive Reviews
Number of Occurrences
0
20
40
60
80
100
120
tommy lee jones
world war ii
close encounters kind
red nosed reindeer
12 year old
rudolph red nosed
does great job
highly recommend movie
new york city
did great job
saving private ryan
does good job
based true story
sci fi movie
sci fi movies
did good job
165 eaton place
sci fi films
lee jones smith
blu ray disc
Top 20 Trigrams




- In the first graph, plotted, I graphed the top 20 bigrams in positive reviews. The most common pair of words that was used throughout the positive reviews, was “blu ray”. Similarly, in the “Top 20 Trigrams in Negative Reviews” graph, “blu ray player” was the most frequent. Blu-ray is known as a type of optical disc format that is used for storing high-definition video and

audio. Essentially, it is a better, more enhanced version of DVDs. The reason why I think it is mentioned so often in the reviews, regardless of if they were positive or negative, is that potentially the reviewers watched the film in a blu ray format. So, they would have used the term "blu ray" to describe their experience of watching the movie.

- Also, by looking at the top 20 bigrams and trigrams for negative reviews, I can see that "mel gibson", "gibson passion christ", and "mel gibson passion" are frequently mentioned. "mel gibson" is also mentioned pretty frequently in the top 20 bigrams in positive reviews graph. I think that these reviews are referring to a writer named "mel gibson", and her film called "The Passion of the Christ". I think that this film specifically did not have very god reviews, because even though it is mentioned in the negative reviews, I don't see it being mentioned in the positive reviews very often, despite the writer still being mentioned in both.
- I also predict that "waste of time and money" along with "don't waste time" were common phrases in negative reviews. Also, the reviewers often mentioned the term "starship troopers" very often using quotes in the negative reviews.
- The bigram "special effects" is also pretty often in both negative and positive reviews. This is because some watchers may have loved the special effects of the movie they watched, whereas others may have felt that they weren't that impressive.

### 1.4.7 Exercise 7 (8 marks):

Throughout the above search for informative words, we have seen that unigrams are not enough, but important words (such as "good") are not always next to the informative words that they describe. Devise a method to extract these informative words. Provide a brief description of how you will extract the informative words. (You do not need to implement your method.)

**Answer.**

# 2 finish

### 2.0.1 Exercise 8 (9 marks):

Write a function(s) that transforms a document into a list of adjective-noun pairs. - For each sentence in the document, find each adjective in the sentence and find the first noun that follows it in the sentence. Combine the adjective with the noun in a string that has the format "adjective noun". - For example, the document "That was a good, long movie" should return `["good movie", "long movie"]`. If no nouns appear after an adjective, do not add that adjective to the output list. - Demonstrate your function works on the first review in `short_movies_tv` and the sentence "The big black dog scared the red cat."

**Answer.**

[939]:
```
def extract_adj_noun_pairs(document):
    # create list to store pairs
    pairs = []
    # convert the document into sentences
   # sentences = sent_tokenize(document)

    # loop through each sentence
```

```
#   for sentence in sentences:
    words = word_tokenize(document)  # Tokenize words in each sentence
    tagged_words = pos_tag(words)  # Perform Part-of-Speech tagging

    adjectives = []  # List to store encountered adjectives

    for i in range(len(tagged_words)):
        word, pos = tagged_words[i]

        if pos.startswith('JJ'):  # Check if the word is an adjective
            adjectives.append(word)
        elif pos.startswith('NN') and adjectives:  # Check if the word is a␣
↪noun following an adjective
            noun = word
            for adj in adjectives:
                pairs.append(f"{adj} {noun}")
            adjectives = []  # Reset adjectives list
      # If no nouns appear after an adjective, do not add that adjective to␣
↪the output list. (do nothing basically)
    return pairs


print("\nSentence:")
print("That was a good, long movie\n")
document2 = "That was a good, long movie"
pairs = extract_adj_noun_pairs(document2)
print(pairs)

print("\nSentence:")
print("The big black dog scared the red cat.\n")
document1 = "The big black dog scared the red cat."
pairs = extract_adj_noun_pairs(document1)
print(pairs)

print("\n\nFirst review in short_movies_tv:")
print(all_reviews[0])
print("\n")
pairs = extract_adj_noun_pairs(all_reviews[0])
print(pairs)
```

```
Sentence:
That was a good, long movie

['good movie', 'long movie']

Sentence:
```

```
The big black dog scared the red cat.

['big dog', 'black dog', 'red cat']


First review in short_movies_tv:
This is a charming version of the classic Dicken's tale.  Henry Winkler makes a
good showing as the "Scrooge" character.  Even though you know what will happen
this version has enough of a change to make it better that average.  If you love
A Christmas Carol in any version, then you will love this.


['charming version', 'classic Dicken', 'good showing', 'better average']
```